How to call a python function from OO Calc

DP 8 Dec 2010
someonesdad1@gmail.com

I don't care for using Open Office's Basic macro language. I would love to be able to call python functions from within a cell formula, as python is a powerful and easy-to-use programming language. Alas, the bad news is this isn't possible. The good news, however, is that with a little work using the Basic macro language, you can call python functions to do the heavy lifting. Some associated bad news is that there is no built-in debugger or IDE for python code, so you write your python scripts in a text editor and figure out what went wrong from the error messages.

The example I'm going to present here is simple but practical. Here's what will happen. We'll call a Basic macro with the contents of two cells: `SI(a1; a2)`. This Basic macro will 1) call the python function `ToSI()` with these two parameters and 2) return the value the python script returns to the spreadsheet cell it was called from. Cell `a1` is a number and cell `a2` is a string. Our python function will examine the first non-space character of the string in `a2` and, if it is a valid SI prefix letter, will return the number in `a1` divided by the appropriate power of 10. If the first character is not a valid SI prefix, the number is returned unchanged.

I use this functionality in some spreadsheets, as it makes it easy to support engineering notation.

The python code

The python code to do this is (save it in a file named `myscript.py`):

```
def ToSI(value, unit):
    '''Expects value to be a number and unit to be a string of zero or more
    characters.  The first non-space character of unit is gotten and, if it
    is a valid SI prefix, the corresponding power of 10 is gotten and value
    is divided by this power and returned as a floating point number.
    Otherwise, value is returned unchanged.
    '''
    eng = {"y": -24, "z": -21, "a": -18, "f": -15, "p": -12, "n": -9, "u": -6,
           "m": -3, "c": -2, "d": -1, "h": 2, "k": 3, "M": 6, "G": 9, "T": 12,
           "P": 15, "E": 18, "Z": 21, "Y": 24}
    unit = unit.strip()
    if not unit or unit[0] not in eng:
        return value
    else:
        return value/float(10**eng[unit[0]])
```

For production code, we'd put in more error checking. For example, we'd test the types of the parameters and return a string indicating an error if they were improper to help the user debug what was wrong. But this isn't needed to show the principles.

This python script can be put in the following places (at least this is my understanding). First, it could be included in the directory containing the python scripts that come with OO[1]. This is probably a bad idea unless you want to deploy there so all users on the computer have access to the macro. Second, the typical way to deploy it is in your "personal" script directory. On Windows, this is (I use forward slashes because I'm in cygwin, but of course if you're in a DOS shell, use backslashes)

```
c:/Documents and Settings/<user>/Application Data/OpenOffice.org/3/user/Scripts
/python
```

and on Unix it's something like

```
~/.openoffice.org.3.0/user/Scripts/python
```

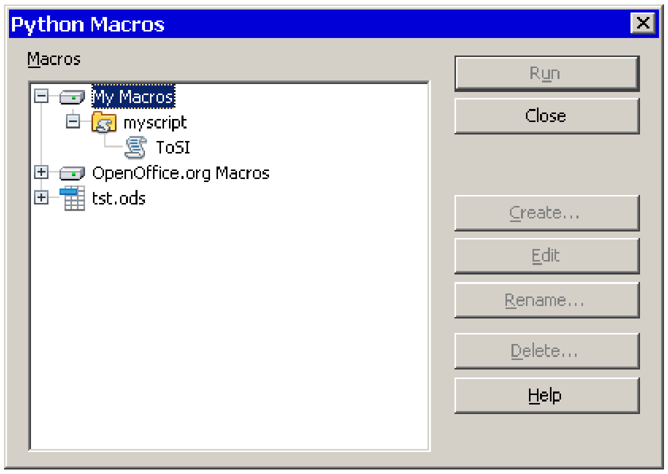
1 There's also a way to deploy multiple scripts in a .pkg file; I refer you to the web for details.

I'm too lazy to boot into a Linux system, so the Unix path could be wrong, but it should be something that rhymes. In both cases, you may have to make the `python` directory. From here on, I'll call this directory your local python script directory.

The third location is to put it into the Calc spreadsheet's file. This looks straightforward; the instructions are here: http://udk.openoffice.org/python/scriptingframework/index.html.

Where you choose to put the python script depends on your deployment model. If you're going to distribute a script to other users on different machines, putting the python script(s) into the spreadsheet file is almost certainly the way to go, as you don't want users having to follow directions to make a directory and copy one or more files there -- someone is likely to do something wrong some day and cause both you and themselves support headaches. If you're the only user of the spreadsheet, then the logical place is your local script directory. From here on, I'll assume that the python script is in your local script directory.

When you use `Tools->Macros->Organize Macros->Python`, you'll get a dialog that looks similar to the following:

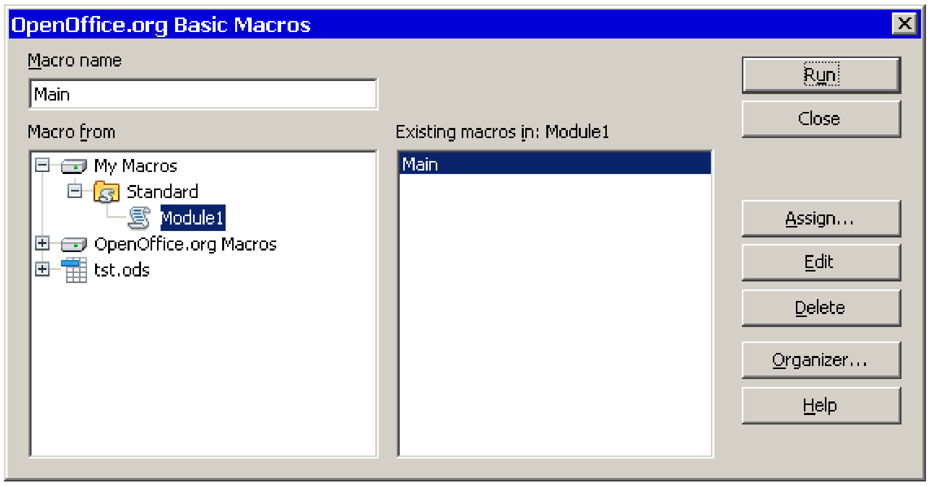


This shows the script files that are in my local python scripts directory; here, there's only one script called `myscript.py` and it contains only one function `ToSI`. It also shows I only have one Open Office file open, the spreadsheet `tst.ods`, where I have the Basic macro that calls the `ToSI()` python function.

OK, we have our python file `myscript.py` saved in our local python script directory. Now we need to write the Basic macro stuff.

The Basic macro

Open a spreadsheet and use `Tools->Macros->Organize Macros->OpenOffice.org Basic` to get the following dialog:



Click on the name of your spreadsheet in the left window, then click on the `New` button to create a new macro module. Then highlight this module name and click on the `Edit` button to bring up the Basic editor. Paste in the following Basic code:

```
' This macro file shows how to use OO Basic to act as an interface to a python
function.

' Keep a global reference to the ScriptProvider because there might be many
calls to
' find a script and we only need to find the ScriptProvider once.
Global MasterScriptProvider

' Specify name and location of python script.
Const URL_Main = "vnd.sun.star.script:myscript.py$"
Const URL_Args = "?language=Python&location=user"

sub Main
      print SI(1.23e-6, "um")
end sub

function SI(a as double, u as string) as double
      script = GetScript("ToSI")
      SI = script.invoke(Array(a, u), Array(), Array())
end function

Function GetMasterScriptProvider()
      dim s as string
      s = "com.sun.star.script.provider.MasterScriptProviderFactory"
      if NOT isObject(MasterScriptProvider) then
            factory = createUnoService(s)
            MasterScriptProvider = factory.createScriptProvider("")
      endif
      GetMasterScriptProvider = MasterScriptProvider
End Function

function GetScript(name$)
      url = URL_Main & name$ & URL_Args
      msp = GetMasterScriptProvider()
      GetScript = msp.getScript(url)
end function
```

This code will perhaps look a bit intimidating, but for now just look at the `SI()` function. This is our Basic macro that calls the python function `ToSI()`. I'm not going to explain the code, as you can research this stuff on the web if you want more details (besides, I don't know much about it).

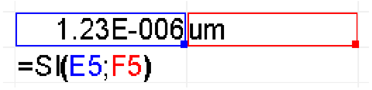
The `GetScript()` function gets a reference to the python function we want to call and the `invoke()` method on this reference object is used to call the python function.

The function parameters we want to pass to the python function are "marshalled" by the `Array()` function. This is a Basic function that creates a (possibly heterogeneous) array object from its parameters. The Basic framework is doing a fair bit of magic behind the scenes to allow you to call into another programming language interface. All we have to know here is that it works.

One other thing is the `Main` function. This is how I test to see whether the Basic macro function SI is working correctly. This lets me click on the toolbar button to run the macro (or just press F5) and I can insert a breakpoint to step through and see what's happening.

Use in the spreadsheet

The final step is to call the Basic macro from our spreadsheet. Here's how I did it in my spreadsheet:



1.23E-006	um
=SI(E5;F5)	

When this function is executed, the resultant in the cell is 1.23, which is the floating point value I expected.

Hopefully, this gives you enough information to use python functions in your Calc spreadsheets.